Titanic Prediction Project

using Machine Learning

BY-PRIYANK KUMAR

Introduction

- ***Titanic*** was a British passenger liner operated by the white star line that sank in the north Atlantic ocean in the early morning hours of 15 April 1912, after striking an iceberg during her maiden voyage from Southampton to New York city. Of the estimated 2200 passenger and crew members aboard, more than 1,500 died, making the sinking one of modern history's deadliest peacetime commercial marine disasters. *Titanic* was the largest ship at the time she entered service and was the second of three *Olympic-class ocean liners* operated by the White Star Line. She was built by the Harland and wolf shipyard in Belfast.

Objective

- The objective of this project is make predictions for who survived and who did not.
- Using the tools of machine learning and various models of machine learning a classification model is to be created to successfully predict who survived and who did not.

Importing The Libraries

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
```

In [5]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

Importing The Dataset

```
In [6]: df=pd.read_csv('titanic.csv')
```

```
In [7]: df.head()  #df.head only shows top part of the dataset it shows by default top 5 rows entering a no. in the bracket wil
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
In [8]: df.shape # no of rows and columns including target column
```

```
Out[8]: (891, 12)
```

```
In [9]: df.columns # displaying only the columns
```

```
Out[9]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
               'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
              dtype='object')
```

The Titanic dataset-Predicting who Survived and who did not

- We will be predicting who survived and who did not by analyzing various models and making relations between them to successfully predict who survived and who did not.
- As we can see in the dataset that Name and Passenger-id of a does not help in predicting the if the person will survive or not there in pre analysis of the data and making the final prediction models therefore we will not be considering these columns.
- Sex, Age, Embarked, SibSp, Fare, Pclass columns can help in predicting that if a person will survive or not so these columns will be taken into consideration while making pre analysis and final predicting mode.

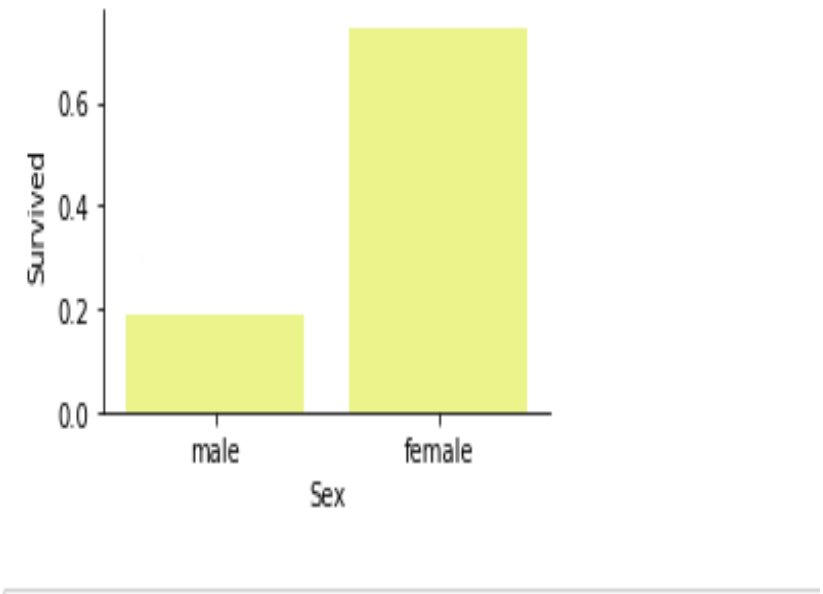
Pre-Prediction by Visualizing the data

- We will make some pre predictions by visualizing the data.
- We will visualize the data in forms of graphs and try to read them and make some pre predictions.

Sex vs Survived

- We can analyze from the graph that females had a greater survival rate than males.

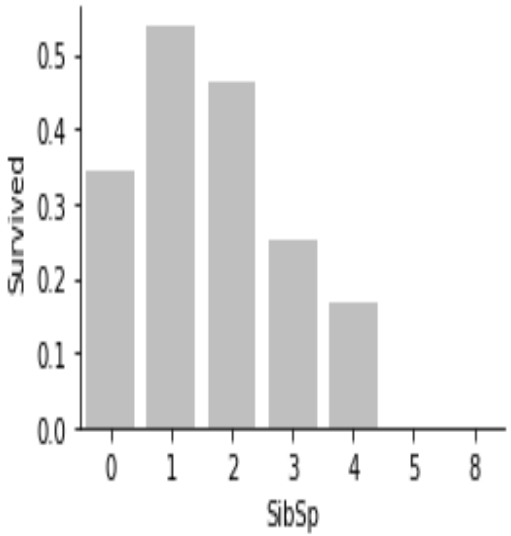
```
Out[147]: <seaborn.axisgrid.FacetGrid at 0x1a23004c90>
```



No. of siblings vs Survived

- We can analyze from the graph that Smaller family size had a greater survival rate than bigger family size.
- Family size of 5-8 siblings had a survival rate of zero.

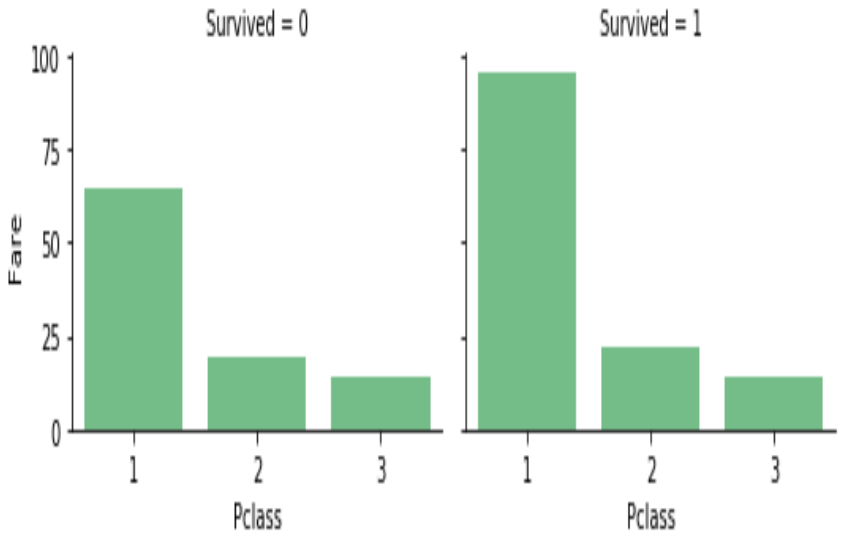
Out[148]: <seaborn.axisgrid.FacetGrid at 0x1a22ff05d0>



Passenger class vs Fare w.r.t Survived

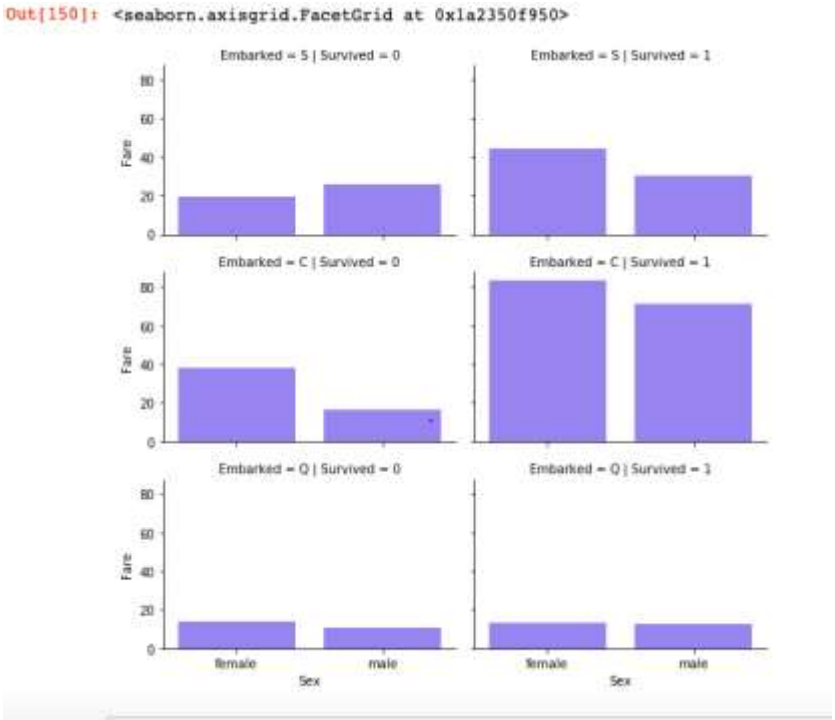
- We can analyze from the graph that Pclass 1 had the highest survival rate out of the three Pclasses and Pclass 3 had the lowest survival rate
- We can also analyze that Passengers who paid the more fare are in Pclass 1
- Passengers who paid more fare also had a higher survival rate than those passenger who paid a lesser fare

Out[132]: <seaborn.axisgrid.FacetGrid at 0x1a21c42550>



Embarked vs Fare

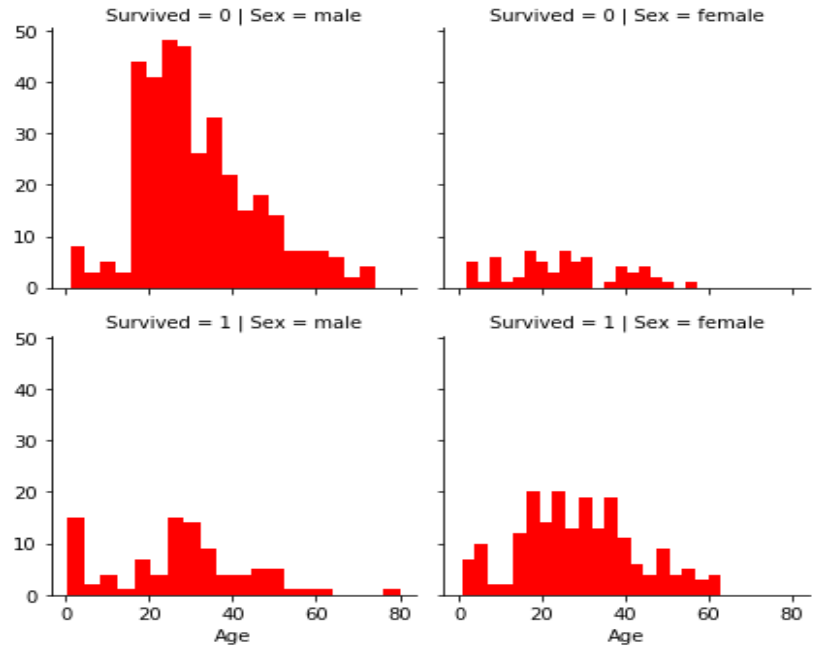
- We can analyze from the graph passengers who embarked from location c had a better survival rate than passengers who embarked from other locations
- We can also analyze from the graph that people who paid a higher fare had a better survival rate than those who paid a lesser fare
- We can also analyze females survival rate is higher than males



Age vs Survived

- We can analyze from the graph that infants with age less than four had a very high survival rate
- Most of passengers were in the age gap 20-40 years
- Lot of passenger in the age gap 20-60 did not survive
- Males in the age gap 20-40 had a lesser survival rate but females in the same age gap had a higher survival rate
- eldest passenger of the ship survived

Out[183]: <seaborn.axisgrid.FacetGrid at 0x1a2ab662d0>



Analyzing Passengers survived

- In total 549 passenger died and 342 survived
- Female had .74 survival rate and males had .18 survival rate
- Amount of males survived were 109 and 468 did not
- Amount of females survived were 232

```
In [10]: df.Survived.value_counts() #no of unique columns so 549 people died and 342 p

Out[10]: 0    549
         1    342
         Name: Survived, dtype: int64
```

```
In [11]: df[['Sex','Survived']].groupby('Sex',as_index=False).mean() #as_index=false i
```

Out[11]:

	Sex	Survived
0	female	0.742038
1	male	0.188908

```
In [12]: df[df.Sex=='male'].Survived.value_counts()

Out[12]: 0    468
         1    109
         Name: Survived, dtype: int64
```

```
In [13]: df[df.Sex=='female'].Survived.value_counts().sum()

Out[13]: 314
```

Analyzing Fare and SibSp

- Passengers with 5 siblings or higher had a zero survival rate
- Only 1 Passenger who paid zero fare survived rest 14 did not

```
In [129]: df[['SibSp','Survived']].groupby('SibSp',as_index=False).mean()
```

Out[129]:

	SibSp	Survived
0	0	0.345395
1	1	0.535885
2	2	0.464286
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

```
In [20]: df[df.Fare==0.0].Survived.value_counts() #passenger who paid the amount zero , only 1 person survived
```

```
Out[20]: 0    14
         1     1
         Name: Survived, dtype: int64
```

Analyzing Embarked and Passenger class

- 644 passengers embarked from S,168 from C and 77 from Q.
- Most passengers embarked from S.
- Passengers who embarked from C had the highest survival rate.
- There were 3 passenger class
- With Pclass 3 had the highest no. of passengers

```
In [22]: df.Pclass.value_counts()

Out[22]: 3    491
         1    216
         2    184
         Name: Pclass, dtype: int64
```

```
In [23]: df.Embarked.value_counts() # in the following lines values of the amount

Out[23]: S    644
         C    168
         Q     77
         Name: Embarked, dtype: int64

In [24]: df[df.Embarked=='C'].Survived.value_counts()

Out[24]: 1    93
         0    75
         Name: Survived, dtype: int64

In [25]: df[df.Embarked=='Q'].Survived.value_counts()

Out[25]: 0    47
         1    30
         Name: Survived, dtype: int64

In [26]: df[df.Embarked=='S'].Survived.value_counts()

Out[26]: 0    427
         1    217
         Name: Survived, dtype: int64

In [27]: df[['Embarked','Survived']].groupby('Embarked',as_index=False).mean()

Out[27]:
```

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.336957

Cleaning the model

```
In [28]: df1=df[['Pclass','Sex','Age','Fare','Embarked']]
```

```
In [29]: df1
```

Out[29]:

	Pclass	Sex	Age	Fare	Embarked
0	3	male	22.0	7.2500	S
1	1	female	38.0	71.2833	C
2	3	female	26.0	7.9250	S
3	1	female	35.0	53.1000	S
4	3	male	35.0	8.0500	S
...	...	...	...	...	...
886	2	male	27.0	13.0000	S
887	1	female	19.0	30.0000	S
888	3	female	NaN	23.4500	S
889	1	male	26.0	30.0000	C
890	3	male	32.0	7.7500	Q

891 rows × 5 columns

df1.isnull().sum() # to find the missing values if sum is notu used then all the values will be displayed

```
In [33]: df2=pd.get_dummies(df1[['Sex','Embarked']]) # get dumm
```

```
In [34]: df2.head()
```

Out[34]:

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	0	1	0	0	1
1	1	0	1	0	0
2	1	0	0	0	1
3	1	0	0	0	1
4	0	1	0	0	1

```
In [35]: df3 = df1.select_dtypes(exclude=['object']) # all the
```

```
In [36]: df3.head()
```

Out[36]:

	Pclass	Age	Fare
0	3	22.0	7.2500
1	1	38.0	71.2833
2	3	26.0	7.9250
3	1	35.0	53.1000
4	3	35.0	8.0500

Cleaning the model

- Df1 is created from the main df which only contains columns which will help predict the survival rate
- Df2 contains those columns which contain subtypes
- Df3 contain those columns which do not have any subtypes
- Final_data is the concatenation of df2 and df3 and we will use final_data to create the models

```
In [37]: # creating the final data
         final_data = pd.concat((df2,df3),axis=1)
         # axis=1: for concatenating 2 or more dataframe as a column wise
```

```
In [38]: final_data.head()
         #various columns have been removed like name,passenger id,ticket no because t
```

Out[38]:

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Pclass	Age	Fare
0	0	1	0	0	1	3	22.0	7.2500
1	1	0	1	0	0	1	38.0	71.2833
2	1	0	0	0	1	3	26.0	7.9250
3	1	0	0	0	1	1	35.0	53.1000
4	0	1	0	0	1	3	35.0	8.0500

Four Models To Predict Survival Rate

- Logistic Regression
- Random Forest Classifier
- Support Vector Classifier
- K Nearest Neighbors

Logistic Regression

- Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

Logistic Regression Implementation

Logictic Regression

In [44]:
```
LG_classifier=LogisticRegression(C=10)
```

In [45]:
```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [46]:
```
x_train
```

Out[46]:
```
array([[-0.73334642,  0.73334642, -0.47927695, ...,  0.83659324,
        -0.02810499, -0.18801432],
       [ 1.36361202, -1.36361202,  2.08647633, ..., -1.55559305,
        -0.00541218,  0.53969044],
       [ 1.36361202, -1.36361202, -0.47927695, ..., -0.35949991,
         0.29627125, -0.46350293],
       ...,
       [-0.73334642,  0.73334642, -0.47927695, ..., -0.35949991,
        -0.6841999 ,  0.89773487],
       [ 1.36361202, -1.36361202, -0.47927695, ...,  0.83659324,
        -0.02810499, -0.52724343],
       [-0.73334642,  0.73334642, -0.47927695, ...,  0.83659324,
        -0.6841999 , -0.51643995]])
```

In [47]:
```
LG_classifier.fit(x_train,y_train)#accuracy of the model can be varied by changing the hyperparameters
```

Out[47]:
```
LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [48]:
```
y_pred1 = LG_classifier.predict(x_test)
```

In [49]:
```
y_pred1
```

Out[49]:
```
array([1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0,
       0, 0, 1])
```

In [50]:
```
cm1 = confusion_matrix(y_test, y_pred1)
print(cm1)
```

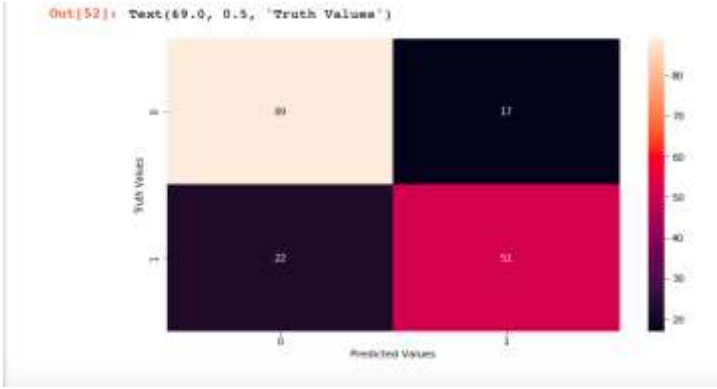
```
[[89 17]
 [22 51]]
```

In [51]:
```
print ("Accuracy : ", accuracy_score(y_test, y_pred1))
```

```
Accuracy :  0.7821229050279329
```

Observation

- After implementing Logistic Regression we observe that it has 78.21% accuracy of successfully predicting the survival rate.
- Observing the confusion matrix of this model we observe that it predicts 89 test values correct and 17 wrong and 51 predict values correct and 21 wrong.
- On varying the hyper parameters like C, penalty we can achieve a better result but it has to be done manually for this model at c=10,penelaty=l2 we got the best result.
- Image shows a heat map of the confusion matrix for a better understanding.



Random Forrest

- Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.
- The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:
- A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

Random Forest Implementation

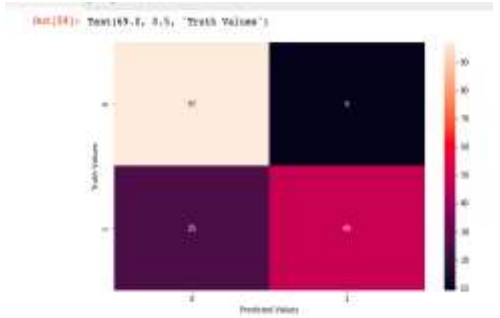
RandomForest Classifier

```
In [53]: RF_classifier=RandomForestClassifier(n_estimators=100)

In [54]: RF_classifier.fit(x_train, y_train)

Out[54]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                criterion='gini', max_depth=None, max_features='auto',
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=100,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)

In [55]: y_pred2= RF_classifier.predict(x_test)

In [56]: y_pred2

Out[56]: array([1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
                1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
                1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
                0, 0, 1])
```

```
In [103]: cm2 = confusion_matrix(y_test, y_pred2)
          print(cm2)

          [[99  7]
           [26 47]]

In [104]: score2=accuracy_score(y_test, y_pred2)
          print('Accuracy :',score2)

          Accuracy : 0.8156424581005587

In [105]: plt.figure(figsize=(10,6))
```

Observation

- After implementing Logistic Regression we observe that it has 81.56% accuracy of successfully predicting the survival rate.
- Observing the confusion matrix of this model we observe that it predicts 99 test values correct and 7 wrong and 47 predict values correct and 26 wrong.
- On varying the hyper parameters like n_estimators we can achieve a better result but it has to be done manually for this model at n_estimators=100 we got the best result.
- We observe that random forest prediction is better than logistic regression model it does a far better job in predicting the survival rate more precisely as noticed in the confusion matrix
- Image shows a heat map of the confusion matrix for a better understanding.



Support Vector Machines(svm)

- Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.
- An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).

SVC Implementation

SupportVector

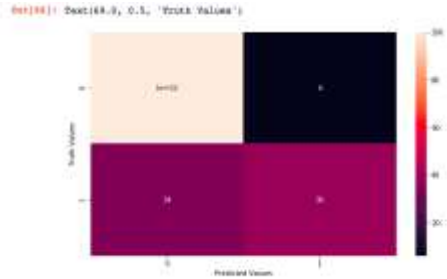
```
In [60]: SV_classifier=SVC()

In [61]: SV_classifier.fit(x_train,y_train)

Out[61]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)

In [62]: y_pred3=SV_classifier.predict(x_test)

In [63]: y_pred3

Out[63]: array([1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
                1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
                1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
                0, 0, 1])
```

```
In [64]: cm3 = confusion_matrix(y_test, y_pred3)
         print(cm3)

[[100   6]
 [ 34  39]]
```

```
In [65]: score3=accuracy_score(y_test, y_pred3)
         print('Accuracy :',score3)

Accuracy : 0.776536312849162
```

Observations

- After implementing Support vector we observe that it has 77.65% accuracy of successfully predicting the survival rate.
- Observing the confusion matrix of this model we observe that it predicts 100 test values correct and 6 wrong and 39 predict values correct and 34 wrong.
- On varying the hyper parameters like C,class_weight we can achieve a better result but it has to be done manually for this model at C=1.0, class_weight=None we got the best result.
- We observe that SVC prediction is worse than both random forest and logistic regression model it does a better job in predicting the test values but overall the survival rate is less off this model from both the previous models
- Image shows a heat map of the confusion matrix for a better understanding



K-nearest neighbors (KNN)

- K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry.
- K-nearest neighbors (KNN) algorithm uses ‘feature similarity’ to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set

KNN Implementation

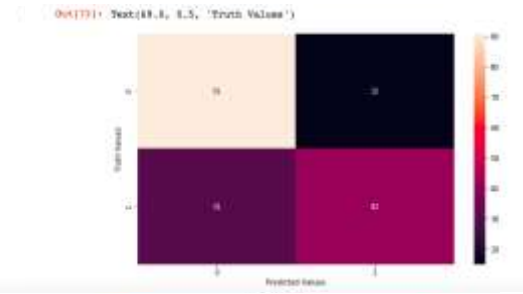
KNeighborsClassifier

```
In [67]: KN_classifier=KNeighborsClassifier()

In [68]: KN_classifier.fit(x_train, y_train)
         y_pred4 = KN_classifier.predict(x_test)

In [69]: y_pred4

Out[69]: array([1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
                1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
                1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
                0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0,
                0, 0, 1])
```

```
In [112]: cm4 = confusion_matrix(y_test, y_pred4)
          print(cm4)

          [[91 15]
           [31 42]]

In [113]: score4=accuracy_score(y_test, y_pred4)
          print('Accuracy :',score4)

          Accuracy : 0.7430167597765364
```

Observation

- After implementing KNN we observe that it has 74.301% accuracy of successfully predicting the survival rate.
- Observing the confusion matrix of this model we observe that it predicts 91 test values correct and 15 wrong and 42 predict values correct and 31 wrong.
- On varying the hyper parameters like n_neighbors we can achieve a better result but it has to be done manually for this model at n_neighbors=21 we got the best result.
- We observe that KNN prediction is worst of all four models it predicts the least accuracy score of the 4 models in predicting survival rate as also observed in the confusion matrix
- Image shows a heat map of the confusion matrix for a better understanding



Final model

- As observed in the table **Random Forest Classifier** is the best model out of the 4 models that is logictic regression, Random Forest Classifier, Support Vector Machine , KNN
- It has the highest accuracy score of 81.56% for predicting the survival rate
- KNN is the worst model out of all these 4 models with the least accuracy score of 74.30%

Out[108]:

	Model	Score
1	Random Forest	0.815642
0	Logistic Regression	0.782123
2	SVC	0.776536
3	KNN	0.743017

Conclusion

- By understanding the models we conclude that the follow pre analysis of the data which were correct-:
- Survival rate of females were more than males
- Infant survival rate is higher than survival rate of elder passengers
- Passengers who paid a higher fare had more survival rate than people who paid lesser fare
- Passenger class 1 had the highest survival rate out of all the passenger classes
- Passengers who embarked from C had the highest survival rate
- Middle ages male has a very less survival rate where as middle aged females survival rate was higher

THANK YOU